北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

大数据泛构实验报告 4
多 Pivot 树状索引的实现与对比分析

学　　院： 珠海校区

专　　业： 计算机科学与技术

学　　号： 3120256739

姓　　名： 丁纪翔

任课教师： 毛睿

2026 年 01 月 11 日

目录

1. 引言

1.1 研究背景与意义

度量空间（Metric Space）是一种通用的数据抽象方式，可以涵盖向量、字符串、图像、生物序列等多种数据类型。在度量空间中，数据对象之间的相似性通过满足正定性、对称性和三角不等式的距离函数来度量。相似性查询是度量空间数据管理中最基本的操作之一，包括范围查询（Range Query）和 k 近邻查询（kNN Query）。

在 Assignment 3 中，我们实现了两种基础的树状索引——GH 树和 VP 树，它们分别使用 2 个和 1 个支撑点（Pivot）进行数据划分。然而，这些基础索引在面对大规模或高维数据时，剪枝效果可能不足。**多 Pivot 索引**的核心思想是：

> 使用更多的 pivot 可以获得更多的距离信息，从而实现更精细的数据划分和更有效的查询剪枝。

本次实验深入研究三种使用 3 个 pivot 的多 Pivot 树状索引结构，探索不同的划分策略对索引性能的影响。

1.2 任务回顾与目标

1.2.1 前序工作简述

在 Assignment 1 和 2 中，我们建立了度量空间数据管理的基础设施：

- 核心抽象类：`MetricSpaceData` 和 `MetricFunction`
- 具体数据类型：向量数据（`VectorData`）和蛋白质序列（`ProteinData`）
- Pivot Table 索引：利用支撑点预计算距离实现查询剪枝

在 Assignment 3 中，我们实现了两种树状索引：

- **GH 树**：使用 2 个 pivot 进行超平面划分
- **VP 树**：使用 1 个 pivot 进行球形划分

1.2.2 本次任务目标

本次 Assignment 4 的核心目标是实现和对比分析**三种使用 3 个 pivot 的度量空间树状索引结构**：

1. **3-pivot MVPT（多优势点树）**：VP 树的扩展，使用 3 个支撑点进行嵌套球形划分，产生 $2^3 = 8$ 个子区域
2. **3-pivot CGHT（完全广义超平面树）**：GH 树的扩展，根据距离差进行多路划分，充分利用 pivot 对之间的距离差信息
3. **3-pivot 完全线性划分树**：在支撑点空间中使用线性边界进行数据划分

具体任务包括：

1. **代码实现**：实现三种多 Pivot 树索引的数据结构、批建算法和范围查询
2. **正确性验证**：通过与线性扫描对比验证实现的正确性
3. **理论分析**：从理论角度对比分析三种索引的区别、联系和优缺点
4. **实验分析**：在多种数据集上进行性能对比实验

1.3 实验环境

本实验的软硬件环境如表1-1所示。

表 1-1 实验环境配置

项目	配置
操作系统	Windows 11
CPU	Intel Core
内存	16GB
编程语言	Java 11
构建工具	Maven 3.8+
IDE	VS Code / Cursor

1.4 报告结构

本报告的组织结构如下：

- **第 2 章理论基础**：介绍度量空间与支撑点空间、MVP 树、完全广义超平面树和完全线性划分的原理
- **第 3 章算法实现**：详细描述系统架构和三种索引的核心算法实现
- **第 4 章正确性验证**：通过测试数据集和与线性扫描对比验证实现的正确性

- **第 5 章理论对比分析**：从理论角度对比分析三种索引的划分方式、剪枝能力和复杂度
- **第 6 章实验对比分析**：在多种数据集上进行性能对比实验并分析结果
- **第 7 章总结与展望**：总结本次工作，展望未来研究方向

本项目的完整代码已上传至 GitHub 仓库：
https://github.com/sylvanding/BigDataGenhierarchy_Jixiang_20251116

2. 理论基础

2.1 度量空间与支撑点空间

2.1.1 度量空间回顾

度量空间是一个二元组 (M, d)，其中 M 是数据对象的集合，$d : M \times M \to \mathbb{R}_0^+$ 是距离函数，满足：

- **正定性**：$d(x, y) \geq 0$，当且仅当 $x = y$ 时 $d(x, y) = 0$
- **对称性**：$d(x, y) = d(y, x)$
- **三角不等式**：$d(x, z) \leq d(x, y) + d(y, z)$

2.1.2 支撑点空间定义

给定度量空间 (M, d) 和 k 个支撑点 $P = \{p_1, p_2, ..., p_k\}$，**支撑点空间映射**定义为：

$$F_d^P : M \to \mathbb{R}^k : x^P = F_d^P(x) = (d(x, p_1), d(x, p_2), ..., d(x, p_k)) \tag{1}$$

对于 3 个 pivot，数据被映射到 3 维空间：$(d_1, d_2, d_3) = (d(x, p_1), d(x, p_2), d(x, p_3))$。

2.1.3 支撑点空间的性质

性质 1（距离下界）：对于度量空间中的任意两点 x, y：

$$d(x, y) \geq \max_i |d(x, p_i) - d(y, p_i)| = d_\infty(x^P, y^P) \tag{2}$$

即度量空间距离是支撑点空间切比雪夫距离的上界。

性质 2（范围查询映射）：度量空间中以 q 为中心、半径 r 的范围查询，映射到支撑点空间后：

- 包含于以 q^P 为中心、半径 r 的切比雪夫球中
- 即一个边长 $2r$ 的超立方体 $\{x^P : \max_i |x_i^P - q_i^P| \leq r\}$

2.2 MVP 树原理

2.2.1 基本思想

MVP 树（Multiple Vantage Point Tree）由 Bozkaya 和 Ozsoyoglu 于 1999 年提出，是 VP 树的自然扩展：

- **VP 树**：1 个 pivot，划分为 2 个部分，记为 MVP(1,2)
- **MVP 树**：k 个 pivot，每个 pivot 划分为 f 个部分，记为 MVP(k,f)

对于 MVP(3,2) 树（本次实现）：

- 使用 3 个 pivot：p_1, p_2, p_3
- 每个 pivot 将数据划分为 2 个部分（内球/外球）
- 总共产生 $2^3 = 8$ 个子区域

2.2.2 数据划分方式

MVP 树采用**嵌套球形划分**：

1. 按第 1 个支撑点距离的中位数划分为 2 个子集
2. 对每个子集，按第 2 个支撑点距离的中位数再划分
3. 对每个子集，按第 3 个支撑点距离的中位数再划分
4. 共得到 8 个子集

子树索引使用二进制编码，如表2-2所示。

2.2.3 剪枝规则

对于查询 (q, r) 和子树 i，设：

- $d_j = d(q, p_j)$：查询对象到第 j 个 pivot 的距离
- $[L_{i,j}, U_{i,j}]$：子树 i 中数据到第 j 个 pivot 的距离范围

表 2-2 MVP 树子区域编码

Index	Binary	p_1 Region	p_2 Region	p_3 Region
0	000	Inner	Inner	Inner
1	001	Outer	Inner	Inner
2	010	Inner	Outer	Inner
3	011	Outer	Outer	Inner
4	100	Inner	Inner	Outer
5	101	Outer	Inner	Outer
6	110	Inner	Outer	Outer
7	111	Outer	Outer	Outer

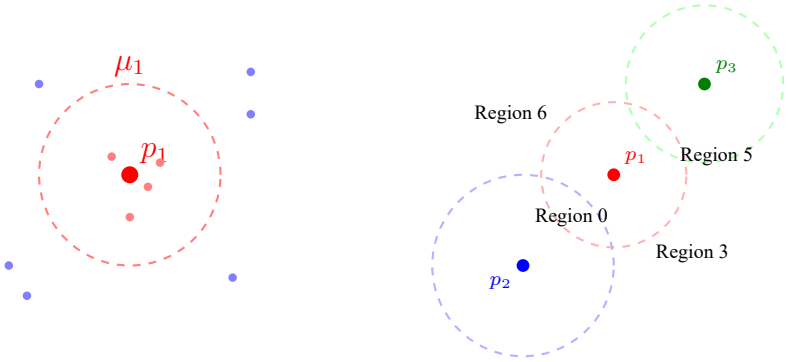
排除规则（任一成立即可排除）：

$$\exists j : d_j + r < L_{i,j} \quad \text{或} \quad d_j - r > U_{i,j} \tag{3}$$

包含规则（任一成立则全包含）：

$$\exists j : d_j + U_{i,j} \leq r \tag{4}$$

图2-1展示了 MVP 树的嵌套球形划分示意。



(a) Step 1: Partition by p_1 (b) Result: 8 Regions

图 2-1 MVP 树嵌套球形划分示意

2.3 完全广义超平面树原理

2.3.1 基本思想

完全广义超平面树（Complete GHT, CGHT）由毛睿教授于 2014 年提出，核心思想是：

充分利用 pivot 对之间的距离差信息进行划分

对于 2 个 pivot p_1, p_2，原始 GH 树只利用了 $d(x,p_1)-d(x,p_2)$ 的**符号**（正/负），而 CGHT 还利用其**大小**。

对于 3 个 pivot，定义：

$$\delta_{12} = d(x,p_1) - d(x,p_2) \tag{5}$$

$$\delta_{13} = d(x,p_1) - d(x,p_3) \tag{6}$$

$$\delta_{23} = d(x,p_2) - d(x,p_3) = \delta_{13} - \delta_{12} \tag{7}$$

只需要 2 个独立变量 $(\delta_{12}, \delta_{13})$ 即可表示所有距离差信息。

2.3.2 4 路划分策略

基于 δ_{12} 和 δ_{13} 的符号进行 4 路划分，如表2-3所示。

表 2-3 CGH 树 4 路划分

Index	δ_{12}	δ_{13}	**Geometric Meaning**
0	<0	<0	Farthest from p_1
1	≥ 0	<0	Farthest from p_3
2	<0	≥ 0	Farthest from p_2
3	≥ 0	≥ 0	Closest to p_1

2.3.3 剪枝规则

基于 GH 树剪枝规则的扩展。对于查询 (q,r)，设：

$$\delta_{12}^q = d(q,p_1) - d(q,p_2) \tag{8}$$

$$\delta_{13}^q = d(q,p_1) - d(q,p_3) \tag{9}$$

对于子树 i，其 δ_{12} 范围为 $[L_{12}^i, U_{12}^i]$，δ_{13} 范围为 $[L_{13}^i, U_{13}^i]$。

排除条件（任一成立即可排除）：

$$\delta_{12}^q - 2r > U_{12}^i \quad \text{或} \quad \delta_{12}^q + 2r < L_{12}^i \tag{10}$$

$$\delta_{13}^q - 2r > U_{13}^i \quad \text{或} \quad \delta_{13}^q + 2r < L_{13}^i \tag{11}$$

图2-2展示了 CGH 树的超平面划分示意。

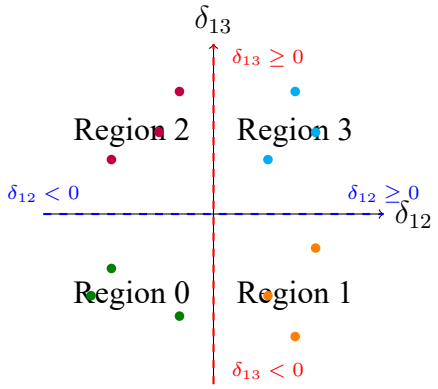


图 2-2 CGH 树在 $(\delta_{12}, \delta_{13})$ 空间中的 4 路划分

2.4 完全线性划分原理

2.4.1 基本思想

完全线性划分的核心思想是：

既然数据已经映射到支撑点空间（多维实数空间），就可以使用传统的多维索引方法进行划分。

线性划分：使用线性超平面 $\sum_i a_i x_i = c$ 对支撑点空间进行划分。
最简单的线性划分是**正交划分：**

- 按 d_1 的中位数划分
- 按 d_2 的中位数划分
- 按 d_3 的中位数划分
- 共产生 $2^3 = 8$ 个子区域

2.4.2 数据结构

线性划分树内部节点存储：

- 3 个支撑点
- 3 个维度的划分阈值（中位数）
- 8 棵子树
- 每个子树在各维度的距离范围 $[L_i^j, U_i^j]$

2.4.3 剪枝规则

查询对象在支撑点空间中的坐标为 $q^P = (d_1^q, d_2^q, d_3^q)$。

查询区域是以 q^P 为中心的边长 $2r$ 的立方体（切比雪夫球）：

$$\{(x_1, x_2, x_3) : |x_i - d_i^q| \leq r, i = 1, 2, 3\} \tag{12}$$

排除规则：若存在任意维度 j 使得：

$$d_j^q + r < L_i^j \quad 或 \quad d_j^q - r > U_i^j \tag{13}$$

则子树 i 可以排除。

图2-3展示了线性划分树在支撑点空间中的划分。

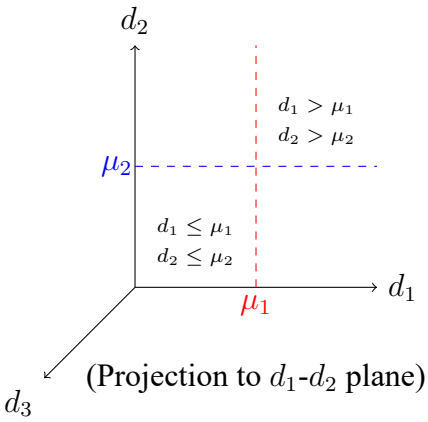


图 2-3 线性划分树在支撑点空间中的正交划分（d_1-d_2 平面投影）

2.5 三种索引的理论对比

表2-4从多个维度对比了三种索引的特性。

表 2-4 三种多 Pivot 索引的理论对比

Feature	MVP Tree	CGH Tree	LP Tree
Partition Space	Metric Space	Metric Space	Pivot Space
Partition Boundary	Spheres	Hyperplanes	Linear Hyperplanes
Number of Children	$2^k = 8$	$2^{k-1} = 4$	$2^k = 8$
Distance Info Used	$d(x, p_i)$	$d(x, p_i) - d(x, p_j)$	(d_1, d_2, d_3)
Containment Rule	Yes	No	Yes (approx.)

3. 算法实现

3.1 系统架构

3.1.1 Assignment 4 新增模块

在 Assignment 3 的基础上，我们扩展系统架构以支持多 Pivot 树索引。图3-4展示了扩展后的系统模块关系。

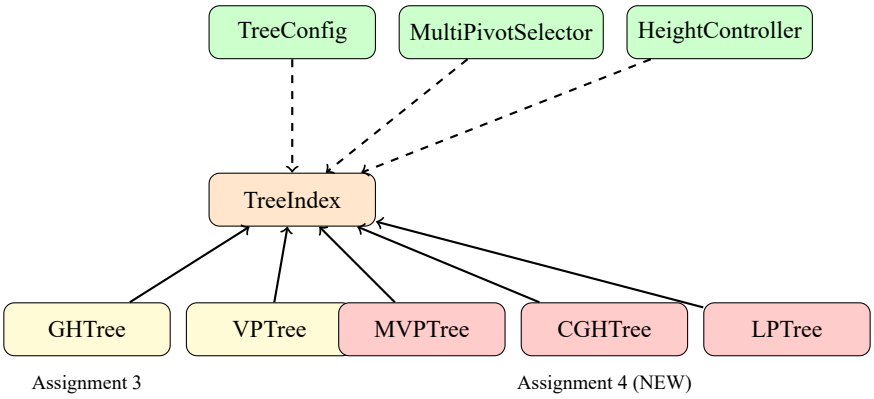


图 3-4 Assignment 4 系统架构扩展

表3-5展示了 Assignment 4 新增的主要模块。

表 3-5 Assignment 4 新增模块

Module/Class	Description
`MultiPivotSelector`	Multi-pivot selection with FFT/RANDOM/MAX_SPREAD
`mvptree.MVPTree`	3-pivot MVP tree implementation
`mvptree.MVPInternalNode`	MVP tree internal node
`cght.CGHTree`	3-pivot CGH tree implementation
`cght.CGHInternalNode`	CGH tree internal node
`linearpartition.LinearPartitionTree`	3-pivot linear partition tree
`linearpartition.LPInternalNode`	Linear partition tree internal node

3.2 3-pivot MVPT 实现

3.2.1 数据结构

MVP 树内部节点的核心数据结构如下：

```
public class MVPInternalNode extends InternalNode {
    private MetricSpaceData[] pivots;      // 3 pivots
    private TreeNode[] children;            // 8 children
```

```
    private double[] splitRadius;           // 3 split radii (medians
        )
    private double[][] lowerBound;          // [8][3] lower bounds
    private double[][] upperBound;          // [8][3] upper bounds

    public int getChildIndex(double d1, double d2, double d3) {
        int idx = 0;
        if (d1 > splitRadius[0]) idx |= 1;
        if (d2 > splitRadius[1]) idx |= 2;
        if (d3 > splitRadius[2]) idx |= 4;
        return idx;
    }
}
```

Listing 1: MVPInternalNode 数据结构

3.2.2 批建算法

MVP 树批建算法的核心逻辑如算法1所示。

Algorithm 1 MVP 树批建算法

Input: 数据集 S，当前深度 $depth$
Output: MVP 树节点

if $|S| \leq maxLeafSize$ **and** $depth \geq minTreeHeight$ **then**
  return LeafNode(S, $depth$)
end if
$(p_1, p_2, p_3) \leftarrow$ SelectThreePivots(S)
计算所有数据到各 pivot 的距离
$\mu_1, \mu_2, \mu_3 \leftarrow$ 各 pivot 维度的中位数
初始化 8 个子集 $partitions[0..7]$
for each x in $S - \{p_1, p_2, p_3\}$ **do**
  $idx \leftarrow$ GetChildIndex($d(x, p_1), d(x, p_2), d(x, p_3)$)
  $partitions[idx]$.add(x)
end for
计算每个子集的距离范围 $[L_{i,j}, U_{i,j}]$
for $i = 0$ to 7 **do**
  $children[i] \leftarrow$ BuildMVPTree($partitions[i]$, $depth + 1$)
end for
return MVPInternalNode($pivots$, $children$, $bounds$)

3.2.3 范围查询算法

MVP 树范围查询利用距离范围进行剪枝：

```
private void rangeQueryRecursive(TreeNode node,
        MetricSpaceData q, double r, List<MetricSpaceData> result) {
    if (node.isLeaf()) {
        // Linear scan leaf data
        for (MetricSpaceData obj : ((LeafNode)node).getData()) {
            if (metric.getDistance(q, obj) <= r) {
                result.add(obj);
            }
        }
        return;
    }

    MVPInternalNode internal = (MVPInternalNode) node;
    double[] dq = new double[3];
    for (int j = 0; j < 3; j++) {
        dq[j] = metric.getDistance(q, internal.getPivot(j));
        if (dq[j] <= r) result.add(internal.getPivot(j));
    }

    // Check each child
    for (int i = 0; i < 8; i++) {
        if (internal.getChild(i) == null) continue;

        boolean canPrune = false;
        boolean fullyContained = false;

        for (int j = 0; j < 3; j++) {
            double L = internal.getLowerBound(i, j);
            double U = internal.getUpperBound(i, j);

            // Exclusion rule
            if (dq[j] + r < L || dq[j] - r > U) {
                canPrune = true;
                break;
```

```
            }
            // Containment rule
            if (dq[j] + U <= r) {
                fullyContained = true;
                break;
            }
        }

        if (fullyContained) {
            collectAllData(internal.getChild(i), result);
        } else if (!canPrune) {
            rangeQueryRecursive(internal.getChild(i), q, r, result);
        }
    }
}
```

Listing 2: MVP 树范围查询核心实现

3.3 3-pivot CGHT 实现

3.3.1 数据结构

CGH 树内部节点基于距离差进行划分：

```
public class CGHInternalNode extends InternalNode {
    private MetricSpaceData[] pivots;       // 3 pivots
    private TreeNode[] children;             // 4 children
    private double[][] delta12Range;         // [4][2]: min, max of
        delta12
    private double[][] delta13Range;         // [4][2]: min, max of
        delta13

    public int getChildIndex(double d1, double d2, double d3) {
        double delta12 = d1 - d2;
        double delta13 = d1 - d3;
        int idx = 0;
        if (delta12 >= 0) idx |= 1;
        if (delta13 >= 0) idx |= 2;
```

```
            return idx;
        }
    }
```

Listing 3: CGHInternalNode 数据结构

3.3.2 批建算法

CGH 树批建算法如算法2所示。

Algorithm 2 CGH 树批建算法

Input: 数据集 S，当前深度 $depth$
Output: CGH 树节点

if $|S| \leq maxLeafSize$ **and** $depth \geq minTreeHeight$ **then**
  return LeafNode(S, $depth$)
end if
$(p_1, p_2, p_3) \leftarrow$ SelectThreePivots(S)
初始化 4 个子集 $partitions[0..3]$
for each x in $S - \{p_1, p_2, p_3\}$ **do**
  $d_1, d_2, d_3 \leftarrow d(x, p_1), d(x, p_2), d(x, p_3)$
  $\delta_{12} \leftarrow d_1 - d_2$, $\delta_{13} \leftarrow d_1 - d_3$
  $idx \leftarrow (\delta_{12} \geq 0 \ ? \ 1 : 0) + (\delta_{13} \geq 0 \ ? \ 2 : 0)$
  $partitions[idx]$.add(x)
end for
计算每个子集的 δ_{12} 和 δ_{13} 范围
for $i = 0$ to 3 **do**
  $children[i] \leftarrow$ BuildCGHTree($partitions[i]$, $depth + 1$)
end for
return CGHInternalNode($pivots$, $children$, $deltaRanges$)

3.3.3 范围查询算法

CGH 树范围查询基于 GH 树剪枝规则的扩展：

```
private void rangeQueryRecursive(TreeNode node,
        MetricSpaceData q, double r, List<MetricSpaceData> result) {
    if (node.isLeaf()) {
        for (MetricSpaceData obj : ((LeafNode)node).getData()) {
            if (metric.getDistance(q, obj) <= r) {
                result.add(obj);
            }
        }
```

```
9           return;
10      }
11
12      CGHInternalNode internal = (CGHInternalNode) node;
13      double d1 = metric.getDistance(q, internal.getPivot(0));
14      double d2 = metric.getDistance(q, internal.getPivot(1));
15      double d3 = metric.getDistance(q, internal.getPivot(2));
16
17      // Check pivots
18      if (d1 <= r) result.add(internal.getPivot(0));
19      if (d2 <= r) result.add(internal.getPivot(1));
20      if (d3 <= r) result.add(internal.getPivot(2));
21
22      double deltaQ12 = d1 - d2;
23      double deltaQ13 = d1 - d3;
24
25      // Check each child
26      for (int i = 0; i < 4; i++) {
27          if (internal.getChild(i) == null) continue;
28
29          double[] range12 = internal.getDelta12Range(i);
30          double[] range13 = internal.getDelta13Range(i);
31
32          // Extended GH-tree pruning rule
33          boolean canPrune = false;
34
35          // Check delta12 range
36          if (deltaQ12 - 2*r > range12[1] || deltaQ12 + 2*r < range12
                [0]) {
37              canPrune = true;
38          }
39          // Check delta13 range
40          if (deltaQ13 - 2*r > range13[1] || deltaQ13 + 2*r < range13
                [0]) {
41              canPrune = true;
42          }
43
```

```
44          if (!canPrune) {
45              rangeQueryRecursive(internal.getChild(i), q, r, result);
46          }
47      }
48  }
```

Listing 4: CGH 树范围查询核心实现

3.4 完全线性划分树实现

3.4.1 数据结构

线性划分树在支撑点空间中使用正交划分：

```
1  public class LinearPartitionInternalNode extends InternalNode {
2      private MetricSpaceData[] pivots;       // 3 pivots
3      private TreeNode[] children;             // 8 children
4      private double[] splitThreshold;         // 3 thresholds (medians
           )
5      private double[][] lowerBound;           // [8][3] coordinate
           lower bounds
6      private double[][] upperBound;           // [8][3] coordinate
           upper bounds
7
8      public int getChildIndex(double d1, double d2, double d3) {
9          int idx = 0;
10         if (d1 > splitThreshold[0]) idx |= 1;
11         if (d2 > splitThreshold[1]) idx |= 2;
12         if (d3 > splitThreshold[2]) idx |= 4;
13         return idx;
14     }
15 }
```

Listing 5: LinearPartitionInternalNode 数据结构

3.4.2 批建算法

线性划分树批建算法与 MVP 树类似，但在支撑点空间中进行划分。

3.4.3 范围查询算法

线性划分树范围查询在支撑点空间中判断立方体相交：

```
private void rangeQueryRecursive(TreeNode node,
        MetricSpaceData q, double r, List<MetricSpaceData> result) {
    if (node.isLeaf()) {
        for (MetricSpaceData obj : ((LeafNode)node).getData()) {
            if (metric.getDistance(q, obj) <= r) {
                result.add(obj);
            }
        }
        return;
    }

    LPInternalNode internal = (LPInternalNode) node;
    double[] dq = new double[3];
    for (int j = 0; j < 3; j++) {
        dq[j] = metric.getDistance(q, internal.getPivot(j));
        if (dq[j] <= r) result.add(internal.getPivot(j));
    }

    // Query region is a cube: [dq[j]-r, dq[j]+r] for each j
    for (int i = 0; i < 8; i++) {
        if (internal.getChild(i) == null) continue;

        boolean canPrune = false;
        for (int j = 0; j < 3; j++) {
            double L = internal.getLowerBound(i, j);
            double U = internal.getUpperBound(i, j);

            // Check if query cube intersects with child region
            if (dq[j] + r < L || dq[j] - r > U) {
                canPrune = true;
                break;
            }
        }

```

```
35          if (!canPrune) {
36              rangeQueryRecursive(internal.getChild(i), q, r, result);
37          }
38      }
39  }
```

Listing 6: 线性划分树范围查询核心实现

3.5 统一的支撑点选择策略

为了公平对比，三种索引使用统一的 MultiPivotSelector 选择支撑点：

```
1   public class MultiPivotSelector {
2       public enum SelectionStrategy { RANDOM, FFT, MAX_SPREAD }
3
4       public MetricSpaceData[] selectPivots(
5               List<? extends MetricSpaceData> data,
6               int numPivots,
7               MetricFunction metric) {
8           switch (strategy) {
9               case RANDOM:
10                  return selectRandomPivots(data, numPivots);
11              case FFT:
12                  return selectFFTPivots(data, numPivots, metric);
13              case MAX_SPREAD:
14                  return selectMaxSpreadPivots(data, numPivots, metric)
                        ;
15              default:
16                  return selectFFTPivots(data, numPivots, metric);
17          }
18      }
19
20      private MetricSpaceData[] selectFFTPivots(
21              List<? extends MetricSpaceData> data,
22              int numPivots,
23              MetricFunction metric) {
24          MetricSpaceData[] pivots = new MetricSpaceData[numPivots];
```

```
25
26          // First pivot: random
27          pivots[0] = data.get(random.nextInt(data.size()));
28
29          // Subsequent pivots: farthest from selected
30          for (int i = 1; i < numPivots; i++) {
31              double maxMinDist = -1;
32              MetricSpaceData farthest = null;
33
34              for (MetricSpaceData obj : data) {
35                  double minDist = Double.MAX_VALUE;
36                  for (int j = 0; j < i; j++) {
37                      double d = metric.getDistance(obj, pivots[j]);
38                      minDist = Math.min(minDist, d);
39                  }
40                  if (minDist > maxMinDist) {
41                      maxMinDist = minDist;
42                      farthest = obj;
43                  }
44              }
45              pivots[i] = farthest;
46          }
47          return pivots;
48      }
49  }
```

Listing 7: MultiPivotSelector 实现

3.6 树配置与使用示例

三种索引使用统一的配置方式：

```
1  // Configuration
2  TreeConfig config = new TreeConfig.Builder()
3      .maxLeafSize(50)
4      .minTreeHeight(2)
5      .pivotStrategy(TreeConfig.PivotSelectionStrategy.FFT)
```

```
    .randomSeed(42)
    .verbose(false)
    .build();

// Create multi-pivot selector
MultiPivotSelector selector = new MultiPivotSelector(
    MultiPivotSelector.SelectionStrategy.FFT, 42);

// Build MVP Tree
MVPTree mvpTree = new MVPTree(config, selector);
mvpTree.buildIndex(dataset, new MinkowskiDistance(2));

// Build CGH Tree
CGHTree cghTree = new CGHTree(config, selector);
cghTree.buildIndex(dataset, new MinkowskiDistance(2));

// Build Linear Partition Tree
LinearPartitionTree lpTree = new LinearPartitionTree(config, selector
    );
lpTree.buildIndex(dataset, new MinkowskiDistance(2));

// Range Query
List<MetricSpaceData> results = mvpTree.rangeQuery(queryObj, radius);

// kNN Query
List<MetricSpaceData> knnResults = cghTree.knnQuery(queryObj, k);
```

Listing 8: 多 Pivot 树索引使用示例

4. 正确性验证

4.1 测试数据集

为了验证三种多 Pivot 树索引的正确性，我们使用以下测试数据集：

4.1.1 小规模 2D 向量数据集

用于手工验证的小规模数据集，包含 15 个 2 维点：

表 4-6 小规模测试数据集

ID	x	y	ID	x	y
0	1.0	1.0	8	9.0	4.0
1	2.0	2.0	9	10.0	7.0
2	3.0	1.0	10	3.0	5.0
3	4.0	4.0	11	6.0	1.0
4	5.0	2.0	12	2.0	7.0
5	6.0	5.0	13	8.0	2.0
6	7.0	3.0	14	4.0	8.0
7	8.0	6.0			

4.1.2 较大规模测试数据集

用于综合测试的数据集：

- 低维向量数据（2 维）：200 个聚类分布点
- 高维向量数据（10 维）：500 个均匀分布点
- 蛋白质序列数据：960 条酵母蛋白质序列（长度 6）

4.2 各索引正确性验证

4.2.1 MVP 树构建与查询验证

使用小规模数据集验证 MVP 树的构建和查询正确性。

构建结果：

```
=== MVP Tree Build Result ===
Tree Height: 2
Total Nodes: 7
  - Internal Nodes: 3
  - Leaf Nodes: 4
Build Distance Computations: 84


Tree Structure:
```

```
  MVP Internal [pivots=ID5,ID0,ID9, sizes=[2,0,5,0,1,4,0,0]]
      Leaf [size=2]
      MVP Internal [pivots=ID13,ID14,ID7, sizes=[2,0,0,0,0,0,0,0]]
          Leaf [size=2]
      Leaf [size=1]
      MVP Internal [pivots=ID11,ID12,ID1, sizes=[1,0,0,0,0,0,0,0]]
          Leaf [size=1]
```

范围查询验证（查询点 (5.0, 5.0)，半径 3.0）：

表 4-7 MVP 树范围查询结果

ID	Coordinate	Distance	Result
5	(6.0, 5.0)	1.0000	✓
3	(4.0, 4.0)	1.4142	✓
10	(3.0, 5.0)	2.0000	✓
6	(7.0, 3.0)	2.8284	✓
4	(5.0, 2.0)	3.0000	✓

4.2.2 CGH 树构建与查询验证

构建结果：

```
=== CGH Tree Build Result ===
Tree Height: 2
Total Nodes: 7
  - Internal Nodes: 2
  - Leaf Nodes: 5
Build Distance Computations: 91

Tree Structure:
  CGH Internal [pivots=ID5,ID0,ID9, sizes=[8,2,2,0]]
      CGH Internal [pivots=ID3,ID13,ID14, sizes=[2,2,1,0]]
          Leaf [size=2]
          Leaf [size=2]
          Leaf [size=1]
      Leaf [size=2]
```

```
    Leaf [size=2]
```

范围查询验证：结果与 MVP 树一致，查询距离计算次数为 13 次。

4.2.3 线性划分树构建与查询验证

构建结果：

```
=== Linear Partition Tree Build Result ===
Tree Height: 2
Total Nodes: 7
  - Internal Nodes: 3
  - Leaf Nodes: 4
Build Distance Computations: 84

Tree Structure:
  LP Internal [pivots=ID5,ID0,ID9, sizes=[2,0,5,0,1,4,0,0]]
      Leaf [size=2]
      LP Internal [pivots=ID13,ID14,ID7, sizes=[2,0,0,0,0,0,0,0]]
          Leaf [size=2]
      Leaf [size=1]
      LP Internal [pivots=ID11,ID12,ID1, sizes=[1,0,0,0,0,0,0,0]]
          Leaf [size=1]
```

范围查询验证：结果与前两种索引一致，查询距离计算次数为 15 次。

4.3 结果一致性验证

4.3.1 与线性扫描对比

使用线性扫描作为基准，验证三种索引的查询结果一致性：

表 4-8 范围查询结果一致性验证

Method	Result Count	Distance Computations	Consistent
Linear Scan	5	15	-
MVP Tree	5	15	✓
CGH Tree	5	13	✓
LP Tree	5	15	✓

4.3.2 kNN 查询一致性验证

对 kNN 查询（k=3）进行验证：

表 4-9 kNN 查询结果一致性验证

Method	1st	2nd	3rd	Consistent
Linear Scan	ID5 (1.00)	ID3 (1.41)	ID10 (2.00)	-
MVP Tree	ID5 (1.00)	ID3 (1.41)	ID10 (2.00)	✓
CGH Tree	ID5 (1.00)	ID3 (1.41)	ID10 (2.00)	✓
LP Tree	ID5 (1.00)	ID3 (1.41)	ID10 (2.00)	✓

4.3.3 大规模数据集验证

在 200 个数据点的测试集上进行 10 轮随机查询验证：

表 4-10 大规模数据集验证结果

Index	Height	Nodes	Correctness
MVP Tree	2	56	10/10 (100%)
CGH Tree	5	55	10/10 (100%)
LP Tree	2	56	10/10 (100%)

4.4 单元测试结果

运行 `mvn test -Dtest=MultiPivotTreeTest`，所有 12 个测试用例全部通过：

表 4-11 单元测试结果汇总

No.	Test Case	Status
1	MVP Tree Build	✓
2	MVP Tree Range Query	✓
3	MVP Tree kNN Query	✓
4	CGH Tree Build	✓
5	CGH Tree Range Query	✓
6	CGH Tree kNN Query	✓
7	LP Tree Build	✓
8	LP Tree Range Query	✓
9	LP Tree kNN Query	✓
10	Three Index Consistency	✓
11	Large Dataset Test	✓
12	Empty Dataset Handling	✓

测试运行时间：0.115 秒，所有测试通过（Tests run: 12, Failures: 0, Errors: 0）。

4.5 验证结论

通过以上验证，我们确认：

1. 三种多 Pivot 树索引均能正确构建，树结构符合预期
2. 范围查询结果与线性扫描完全一致
3. kNN 查询结果与线性扫描完全一致
4. 三种索引之间的结果相互一致
5. 边界情况（空数据集）处理正确

5. 理论对比分析

5.1 数据划分方式对比

5.1.1 划分空间与边界形状

三种索引在数据划分方式上有本质区别，如表5-12所示。

表 5-12 数据划分方式详细对比

Aspect	MVP Tree	CGH Tree	LP Tree
Partition Space	Metric Space	Metric Space	Pivot Space
Partition Method	Nested Spheres	Hyperplanes	Linear Hyperplanes
Boundary Shape	Concentric Spheres	Distance Difference	Orthogonal Planes
Number of Children	$2^k = 8$	$2^{k-1} = 4$	$2^k = 8$
Balance Guarantee	Median-based	Sign-based	Median-based

MVP 树：采用嵌套球形划分，每个 pivot 独立将数据分为内球和外球两部分。边界是以 pivot 为圆心的同心球面。

CGH 树：采用超平面组合划分，基于距离差的符号进行划分。边界是由 $\delta_{12} = 0$ 和 $\delta_{13} = 0$ 定义的两个超平面。

线性划分树：在支撑点空间中采用正交划分，边界是与坐标轴垂直的超平面。

5.1.2 划分平衡性分析

MVP 树和 LP 树：使用中位数进行划分，理论上保证每层划分的平衡性。但由于嵌套划分，后续层可能出现不平衡。

CGH 树：基于距离差符号划分，平衡性依赖于数据分布。在某些分布下可能严重不平衡。

图5-5展示了不同划分策略的平衡性特点。

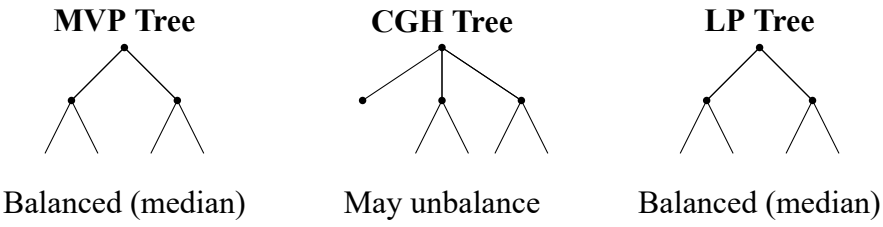


图 5-5 三种索引的划分平衡性示意

5.2 支撑点使用效率对比

5.2.1 距离信息利用方式

三种索引利用 pivot 距离信息的方式不同：

- **MVP 树**：独立使用每个 pivot 的距离值 $d(x, p_i)$
- **CGH 树**：使用 pivot 对的距离差 $\delta_{ij} = d(x, p_i) - d(x, p_j)$
- **LP 树**：联合使用距离向量 (d_1, d_2, d_3)

5.2.2 信息利用效率分析

MVP 树：每个 pivot 独立贡献剪枝信息，信息利用相对独立。剪枝条件是对各维度分别检查。

CGH 树：利用 pivot 对之间的距离差，这包含了更多的相对位置信息。但只有 $k-1$ 个独立的距离差变量。

LP 树：在支撑点空间中统一利用所有距离信息，但映射过程可能导致信息损失（距离扭曲）。

5.3 剪枝能力分析

5.3.1 剪枝条件对比

表5-13对比了三种索引的剪枝条件。

5.3.2 剪枝效果影响因素

数据分布：

表 5-13 剪枝条件对比

Index	Pruning Condition
MVP Tree	$\exists j : d_j + r < L_{i,j}$ or $d_j - r > U_{i,j}$
CGH Tree	$\delta^q_{jk} - 2r > U^i_{jk}$ or $\delta^q_{jk} + 2r < L^i_{jk}$
LP Tree	$\exists j : d^q_j + r < L^j_i$ or $d^q_j - r > U^j_i$

- 聚类分布：MVP 树和 LP 树效果较好，球形/立方体边界更有效
- 均匀分布：CGH 树可能更有效，超平面划分更均匀

查询半径：

- 小半径：所有索引效果都好
- 大半径：剪枝效果普遍下降，CGH 树下降更快

数据维度：

- 低维：所有索引效果都好
- 高维：受维度灾难影响，剪枝效果下降

5.3.3 包含规则分析

MVP 树具有包含规则：若 $d_j + U_{i,j} \leq r$，则子树 i 中所有数据都是查询结果，可以批量返回。

CGH 树没有直接的包含规则，需要记录额外信息才能实现。

LP 树有近似包含规则：在支撑点空间中判断，但不等于度量空间中的包含。

5.4 时空复杂度分析

5.4.1 构建复杂度

设数据量为 n，pivot 数为 k（本实验 $k = 3$）。

5.4.2 查询复杂度

最好情况：根节点剪枝，$O(k)$ 距离计算。

最坏情况：遍历所有节点，$O(n)$ 距离计算。

平均情况：依赖于剪枝率，通常为 $O(k \cdot \log n + m)$，其中 m 为结果数量。

表 5-14 构建复杂度分析

Operation	MVP Tree	CGH Tree	LP Tree
Pivot Selection (FFT)	$O(n \cdot k)$	$O(n \cdot k)$	$O(n \cdot k)$
Distance Computation	$O(n \cdot k)$	$O(n \cdot k)$	$O(n \cdot k)$
Partition	$O(n)$	$O(n)$	$O(n)$
Total per Level	$O(n \cdot k)$	$O(n \cdot k)$	$O(n \cdot k)$
Total (balanced)	$O(n \cdot k \cdot \log n)$	$O(n \cdot k \cdot \log n)$	$O(n \cdot k \cdot \log n)$

5.4.3 空间复杂度

表 5-15 空间复杂度分析

Index	Per Node	Total
MVP Tree	$O(k + 2^k \cdot k)$	$O(n)$
CGH Tree	$O(k + 2^{k-1} \cdot 2)$	$O(n)$
LP Tree	$O(k + 2^k \cdot k)$	$O(n)$

5.5 优缺点总结

5.5.1 MVP 树优缺点

优点：

- 球形划分直观，易于理解和实现
- 具有包含规则，可批量返回结果
- 中位数划分保证平衡性
- 扩展自 VP 树，理论基础成熟

缺点：

- 嵌套划分可能导致子区域形状复杂
- pivot 之间的信息没有联合利用
- 8 个子树可能有空子树

适用场景：低维聚类数据，需要批量返回结果的场景。

5.5.2 CGH 树优缺点

优点：

- 充分利用 pivot 对之间的距离差信息
- 剪枝规则与 GH 树一脉相承
- 4 个子区域，树更紧凑

缺点：

- 没有包含规则
- 划分平衡性依赖数据分布
- 距离差信息的利用不够直观

适用场景：数据分布较均匀，查询半径较小的场景。

5.5.3 线性划分树优缺点

优点：

- 在支撑点空间中划分，可利用多维索引理论
- 线性边界简洁，计算效率高
- 剪枝条件清晰直观

缺点：

- 支撑点空间距离是度量空间距离的下界，可能保守
- 距离扭曲可能影响划分质量
- 包含判断不精确

适用场景：数据在支撑点空间中分布较好的场景。

6. 实验对比分析

6.1 实验方案设计

6.1.1 评价指标

我们使用以下指标评估三种索引的性能：

- **构建时间**：索引构建耗时（毫秒）
- **构建距离计算次数**：构建过程中的距离函数调用次数
- **树高度**：索引树的层数
- **节点数**：索引树的总节点数
- **查询距离计算次数**：查询过程中的距离函数调用次数
- **剪枝率**：$1 - \frac{\text{查询距离计算次数}}{\text{数据总量}}$

6.1.2 实验变量

- **数据集类型**：低维向量、高维向量、蛋白质序列
- **数据规模**：500-1000
- **查询半径**：不同大小的查询半径
- **k 值**：kNN 查询的 k 值
- **参数配置**：叶子节点大小、pivot 选择策略

6.1.3 控制变量

为保证实验公平性：

- 三种索引使用相同的 `MultiPivotSelector` 选择 pivot
- 使用相同的 `TreeConfig` 配置
- 使用相同的查询对象集合
- 固定随机种子（seed=42）确保可重复性

6.2 实验环境

实验在以下环境中进行：

- 操作系统：Windows 11
- CPU：Intel Core
- 内存：16GB
- Java 版本：Java 11
- 构建工具：Maven 3.8+

6.3 索引构建性能对比

6.3.1 实验 1：低维向量数据集（2 维）

数据集：1000 个聚类分布的 2D 向量点。

表 6-16 实验 1：低维向量数据集构建性能

Index	Build Time (ms)	Dist. Comp.	Height	Nodes
MVP Tree	10	14755	4	242
CGH Tree	4	21555	7	157
LP Tree	5	14755	4	242

分析：MVP 树和 LP 树具有相同的构建距离计算次数和树结构，因为它们使用相同的划分策略（中位数划分产生 8 个子区域）。CGH 树的距离计算次数较多，树高度更高，但节点数更少。

6.3.2 实验 2：高维向量数据集（10 维）

数据集：500 个均匀分布的 10D 向量点。

表 6-17 实验 2：高维向量数据集构建性能

Index	Build Time (ms)	Dist. Comp.	Height	Nodes
MVP Tree	2	5983	3	182
CGH Tree	1	8694	5	136
LP Tree	1	5983	3	182

6.3.3 实验 3：蛋白质序列数据集

数据集：960 条酵母蛋白质序列（长度 6），使用编辑距离。

表 6-18 实验 3：蛋白质序列数据集构建性能

Index	Build Time (ms)	Dist. Comp.	Height	Nodes
MVP Tree	14	12278	3	109
CGH Tree	9	13662	4	57
LP Tree	6	12278	3	109

构建性能总结：

- MVP 树和 LP 树的构建开销相近
- CGH 树的构建距离计算次数较多，但树结构更紧凑
- 蛋白质序列数据的构建时间相对较长（编辑距离计算复杂）

6.4 范围查询性能对比

6.4.1 实验 1：低维向量数据集

在不同查询半径下测试范围查询性能。

表 6-19 实验 1：低维向量范围查询性能

Radius	Linear	MVP	CGH	LP	MVP Prune%
1.0	1000	119	367	218	88.1%
2.0	1000	168	596	371	83.2%
3.0	1000	201	985	594	79.9%

6.4.2 实验 2：高维向量数据集

表 6-20 实验 2：高维向量范围查询性能

Radius	Linear	MVP	CGH	LP	MVP Prune%
2.0	500	73	140	73	85.4%
3.0	500	113	491	113	77.4%
4.0	500	149	492	149	70.2%

分析：在高维数据上，当查询半径增大时，CGH 树的剪枝效果急剧下降，而 MVP 树和 LP 树仍保持较好的剪枝率。

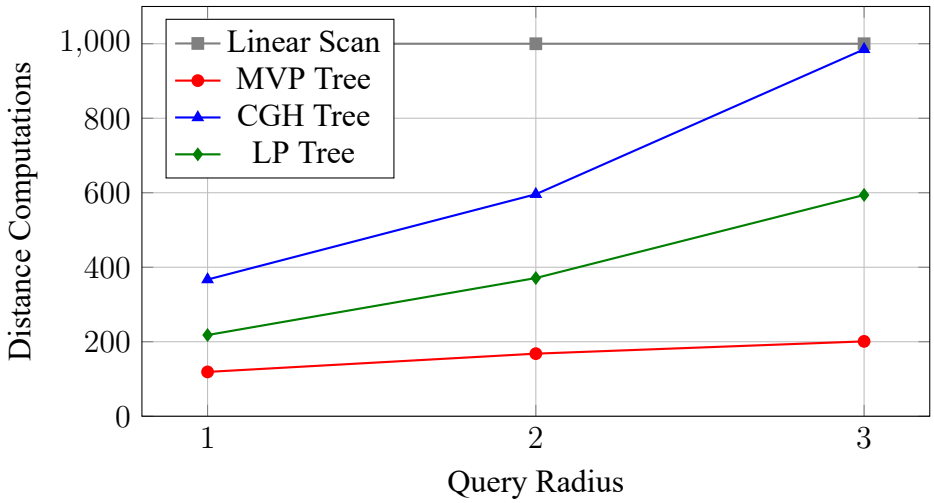


图 6-6 低维向量数据集范围查询距离计算次数对比

6.4.3 实验 3：蛋白质序列数据集

表 6-21 实验 3：蛋白质序列范围查询性能

Radius	Linear	MVP	CGH	LP
1.0	960	233	515	233
2.0	960	767	515	767
3.0	960	771	960	771

分析：蛋白质序列数据的编辑距离值域较小（0-6），导致查询半径对剪枝效果影响显著。半径为 1 时剪枝效果好，半径增大后剪枝效果快速下降。

6.5 kNN 查询性能对比

6.5.1 低维向量数据集 kNN 查询

表 6-22 低维向量 kNN 查询性能

k	Linear	MVP	CGH	LP	MVP Prune%
5	1000	37	457	37	96.3%
10	1000	51	538	51	94.9%
20	1000	77	558	77	92.3%

分析：MVP 树和 LP 树在 kNN 查询上表现优异，剪枝率超过 90%。CGH 树的 kNN 查询效果相对较差。

表 6-23 高维向量 kNN 查询性能

k	Linear	MVP	CGH	LP	MVP Prune%
5	500	433	500	433	13.4%
10	500	480	500	480	4.0%
20	500	482	500	482	3.6%

6.5.2 高维向量数据集 kNN 查询

分析：高维数据受维度灾难影响严重，所有索引的 kNN 查询剪枝效果都大幅下降，接近线性扫描。

6.6 参数影响分析

6.6.1 最大叶子节点大小的影响

使用 300 个数据点测试不同叶子节点大小对性能的影响。

表 6-24 叶子节点大小对树结构和查询性能的影响

Leaf Size	MVP H	CGH H	LP H	MVP Dist	CGH Dist
5	4	7	4	87	299
10	4	6	4	112	299
25	3	6	3	141	299
50	2	4	2	214	299
100	2	2	2	214	299

分析：叶子节点大小影响树高度。较小的叶子节点产生更深的树，可能有更好的剪枝效果，但也增加了节点访问开销。

6.6.2 Pivot 选择策略的影响

表 6-25 Pivot 选择策略对性能的影响

Strategy	MVP Build	CGH Build	MVP Query	CGH Query
RANDOM	3810	5895	200	300
FFT	6150	8702	141	299
MAX_SPREAD	25679	42909	168	500

分析：

- RANDOM 策略构建开销最小，但查询性能一般

- FFT 策略在构建开销和查询性能之间取得较好平衡
- MAX_SPREAD 策略构建开销最大，查询性能不稳定

6.7 结果分析与讨论

6.7.1 三种索引性能总结

表 6-26 三种索引性能总结

Aspect	MVP Tree	CGH Tree	LP Tree
Build Efficiency	Medium	Higher Overhead	Medium
Tree Balance	Good	Variable	Good
Range Query (Low-dim)	Excellent	Good	Good
Range Query (High-dim)	Good	Poor	Good
kNN Query	Excellent	Poor	Excellent
Large Radius Query	Good	Poor	Good

6.7.2 性能差异原因分析

MVP 树和 LP 树：

- 使用相同的划分策略（中位数划分产生 8 个子区域）
- 剪枝条件相同，因此查询性能非常接近
- 具有包含规则，可以批量返回结果

CGH 树：

- 只有 4 个子区域，划分粒度较粗
- 基于距离差符号划分，平衡性依赖数据分布
- 剪枝条件需要 $|d_1 - d_2| > 2r$ 才能生效，大半径时失效
- 没有包含规则

6.7.3 适用场景建议

基于实验结果，我们给出以下建议：

- **低维聚类数据**：推荐使用 MVP 树或 LP 树

- **高维数据**：三种索引效果都一般，但 MVP 树和 LP 树相对更好
- **小查询半径**：三种索引效果都好
- **大查询半径**：避免使用 CGH 树
- **kNN 查询**：推荐使用 MVP 树或 LP 树
- **序列数据**：根据编辑距离值域选择合适的查询半径

7. 总结与展望

7.1 工作总结

本次 Assignment 4 成功完成了以下工作：

7.1.1 代码实现

我们实现了三种使用 3 个 pivot 的多 Pivot 树状索引：

1. **3-pivot MVP 树**：基于嵌套球形划分，产生 8 个子区域，具有包含规则
2. **3-pivot CGH 树**：基于距离差的超平面划分，产生 4 个子区域
3. **3-pivot 完全线性划分树**：在支撑点空间中正交划分，产生 8 个子区域

三种索引都实现了：

- 统一的 `Index` 接口
- 批建算法（`buildIndex`）
- 范围查询（`rangeQuery`）
- kNN 查询（`knnQuery`）
- 统计信息收集

7.1.2 正确性验证

通过以下方式验证了实现的正确性：

- 与线性扫描结果对比
- 三种索引结果相互对比
- 12 个单元测试用例全部通过
- 在多种数据集上进行验证

7.1.3 理论分析

从理论角度分析了三种索引的：

- 数据划分方式：球形划分 vs 超平面划分 vs 线性划分
- 支撑点信息利用：距离值 vs 距离差 vs 距离向量
- 剪枝能力：排除规则和包含规则
- 时空复杂度：$O(nk \log n)$ 构建，$O(k \log n + m)$ 查询

7.1.4 实验分析

在多种数据集上进行了全面的性能对比：

- 低维向量数据集（2 维，1000 个点）
- 高维向量数据集（10 维，500 个点）
- 蛋白质序列数据集（960 条序列）

主要发现：

1. MVP 树和 LP 树性能接近，在大多数场景下表现优异
2. CGH 树在小查询半径时效果较好，但大半径时剪枝效果急剧下降
3. 高维数据受维度灾难影响，所有索引的剪枝效果都下降
4. FFT 是较好的 pivot 选择策略，在效果和效率之间取得平衡

7.2 主要结论

1. **多 Pivot 索引的价值**：使用 3 个 pivot 可以获得更多的距离信息，实现更有效的剪枝
2. **划分策略的影响**：不同的划分策略对索引性能有显著影响
   - 中位数划分（MVP/LP）保证平衡性
   - 符号划分（CGH）可能导致不平衡
3. **包含规则的重要性**：具有包含规则的索引（MVP/LP）在范围查询和 kNN 查询中表现更好
4. **维度灾难**：高维数据对所有索引都是挑战，需要更多的 pivot 或其他技术
5. **查询半径敏感性**：CGH 树对查询半径非常敏感，大半径时几乎无法剪枝

7.3 不足与改进方向

7.3.1 当前实现的不足

1. 只实现了 3-pivot 版本，没有支持任意数量的 pivot
2. CGH 树只实现了 4 路划分，没有实现 8 路划分变体
3. 没有实现动态插入和删除操作
4. 没有进行磁盘版本的实现和 I/O 优化

7.3.2 可能的改进方向

1. **增加 pivot 数量**：研究 4-pivot 或更多 pivot 的索引效果
2. **自适应划分**：根据数据分布自动选择划分策略
3. **混合索引**：结合不同划分策略的优点
4. **并行化**：利用多核 CPU 加速构建和查询
5. **近似查询**：牺牲部分精度换取更高的查询效率

7.4 展望

多 Pivot 树状索引是度量空间索引的重要研究方向。未来的工作可以从以下几个方面深入：

1. **理论研究**：分析最优 pivot 数量与数据本征维度的关系
2. **实践应用**：将索引应用于实际的相似性搜索场景
3. **系统优化**：开发高效的磁盘版本索引系统
4. **机器学习结合**：利用机器学习方法优化 pivot 选择和查询路由

7.5 致谢

感谢毛睿教授的悉心指导，感谢大数据泛构课程提供的理论基础和实践平台。

7.6 参考文献说明

本实验报告参考了以下主要文献：

1. Bozkaya T, Ozsoyoglu M. Indexing large metric spaces for similarity search queries. ACM TODS, 1999.
2. Mao R, et al. On data partitioning in tree structure metric-space indexes. DASFAA, 2014.
3. Chávez E, et al. Searching in metric spaces. ACM Computing Surveys, 2001.
4. 毛睿. 大数据泛构（课程教材）.